JML and ESC/Java2: An Introduction

Karl Meinke
School of Computer Science and Communication, KTH
Java Modeling Language (JML)

A behavioural interface specification language for Java
• Design-by-contract paradigm (Eiffel)
• Lightweight, usable by practising programmers

Add assertions to Java source code, e.g.
• Method preconditions
• Method postconditions
• Class invariants

Assertions are extended Java boolean expressions, added through comments in .java files:

/*@ “my JML stuff” */
//@ ”my JML stuff”
Method Pre- and Postconditions

Wallet Example:
//@ requires amount \geq 0;
//@ ensures balance == old(balance) - amount;
//@ ensures result == balance;
public int debit(int amount) { ... }

old(E): refers to E in the state before the method call
result: refers to the return value
Class Invariants

Wallet Example:

```java
public class Wallet {

    public static final short MAX_WALLET;

    private short balance;

    /*@ invariant 0 <= balance && balance <= MAX_WALLET; */

    ...
}
```

Class invariants must be:

- Preserved by all methods in a class, i.e. implicitly included in both the pre- and postconditions of methods;
- Established by all constructors, i.e. implicitly included in postconditions of constructors
OtherAnnotations

*Assumptions* used but not checked (cf. *math axioms*): e.g.
//@ assume x+y==y+x;

*Exceptional postconditions:*
//@ requires amount == 0;
//@ ensures balance == \old(balance) + amount;
//@ ensures (WalletOverflow) balance == \old(balance) ;

**public int credit(int amount) throws WalletOverflow { ... }**

Only WalletOverflow can be thrown, and whenever it is thrown balance == \old(balance) holds (no action!).

*Assertions* both used and checked: e.g. //@ assert balance == 0;
ESC/Java2

Extended Static Checker originally developed at Compaq Systems Research Center, now an open-source project maintained by Kiniry et al. at University College Dublin.

Features and properties:
• Checks JML-annotated java code automatically or interactively
• **Unsound**: does not find all incorrect annotations
• **Incomplete**: may report nonexistent errors
• Good at routine checks of relatively simple properties
  – null dereferences
  – Array indices out-of-bounds
• Unsuitable for complete program verification
  – Theorem prover built to be automatic
  – Loops only traversed once
ESC/Java2 Benefits

Highlights of static checking:

• Automatic use, only annotation overhead
• Finds more advanced properties than type checkers
• Forces important properties to be explicitly recorded
• Makes it easier to understand and maintain code
• Success stories from experimental use in real-world

Design-by-contract paradigm especially useful when program reliability and security is paramount!
Loop Invariant Example

Consider the following method:

```java
public static int plus(int x, int y) {
    int s = 0;
    int i = x;
    while (i != 0) {
        s = s + y;
        i = i - 1;
    }
    return s;
}
```

A loop invariant is a property that is true
• before the loop
• before and after the loop body

Loop invariants can be checked by ESC/Java2 if the loop_invariant pragma is used.
Loop Invariant Example, Continued

The annotated code:

```java
//@ axiom (\forall int i, j, k; i \times (j - k) == i \times j - i \times k);
//@ requires x >= 0;
//@ ensures \result == y \times x;
public static int plus(int x, int y) {
    int s = 0;
    int i = x;
    //@ loop_invariant s == y \times x - y \times i;
    while (i != 0) {
        s = s + y;
        i = i - 1;
    }
    return s;
}
```

Loop invariant captures what really goes on in the loop!
Further reading

Information about JML and related tools:
http://www.jmlspecs.org
ESCJava2 documentation and downloads:
http://secure.ucd.ie/products/opensource/ESCJava2/
Pointers to information about formal methods:
http://vl.fmnet.info
More examples and links:
http://www.csc.kth.se/~palmskog/2G4514/
Alternatives to JML/ESCJava2 (1)

Alloy: was developed in the hope of adapting first-order logic to allow fully automatic analysis. To do this Alloy sacrifices the ability to totally prove a system's correctness. Rather, it attempts to find counterexamples within a limited scope that violate the constraints of the system.

Alloy was developed by the Software Design Group at MIT. In 1997 they produced the first Alloy prototype, which was then a rather limited object modelling language. Over the years Alloy has developed into a full structural modelling language capable of expressing complex structural constraints and behaviour.
Object Constraint Language (OCL) is the constraint language of UML. It was developed at IBM and ObjecTime Limited and was added to the UML in 1997. Because it was initially designed to be an annotation language for UML class diagrams, it does not include a textual notation for declarations. Variants of OCL such as USE overcome this limitation.

Many tools are available supporting OCL such as Octopus and the Eclipse Model Development Tools. Typical features include the interpretation of OCL constraints over test cases and code generation. Some, such as the USE tool mentioned above, support design-time analysis and allow exhaustive search over a finite space of cases similar to Alloy.
Vienna Development Method (VDM) is a set of techniques for developing computer systems. It originated from IBM's Vienna Laboratory in the mid-1970s.

In 1988 Peter Froome developed a tool called SpecBox which was the first industrialised tool for checking VDM specification. SpecBox is used in civil nuclear, railway and security applications. There are many other tools for checking VDM specification such as IFAD VDM-SL and Centaur-VDM environment which is an interactive and graphical tool for VDM.

However unlike Alloy these do not provide fully automatic analysis in the style of a model checker. Both Alloy and VDM support object-orientation and concurrency.

VDM has been standardized and is still widely used in industry by such organisations as British Aerospace Systems & Equipment, Rolls Royce and Dutch Department of Defence. Alloy is not used as much in industry as VDM.